

Übung 11 – Algorithmen II

Sebastian Lamm, Demian Hespe – *lamm@kit.edu, hespe@kit.edu*
http://algo2.iti.kit.edu/AlgorithmenII_WS18.php

Institut für Theoretische Informatik - Algorithmik II

```
result = current_weight;
return true;

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ) );
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ) );
        Weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ) );
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ) );
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::INCORRECTLY_RELAXED_EDGES ) );
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

Ankündigungen

Vorlesung 4.2.:

- Peter Sanders (Wiederholung aus Sicht der Vorlesung)

Übung 5.2.:

- Demian Hespe, Sebastian Lamm
- Wiederholung / Fragen zur Klausur (Themenwünsche im Forum, per eMail)
- Letztes Übungsblatt ist online

Klausur 19.2.:

- Anmeldung noch bis 12.2. möglich

Themenübersicht

- *in-place Multikey Quicksort*
(Sortierung von Zeichenketten *in-place*)

- Suche mit Hilfe von Suffix-Arrays
 - Wiederholung aus Vorlesung
 - Beschleunigung mittels LCP-Array

in-place Multikey Quicksort

Wiederholung

Bentley, Sedgewick (1997)

(Three-way Radix Quicksort)

- sortiert Elemente mit **mehreren Schlüsseln** wie msd-*Radixsort*
→ z.B. Stellen einer Zahl, Zeichen eines Strings
- für einen Schlüssel wird *Quicksort* mit **drei Fällen** ausgeführt
→ **kleiner als, gleich, größer als** das Pivotelement
- PartitionierungsSchritt kann ***in-place*** erfolgen
→ ähnlich wie bei normalem *Quicksort*

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$

S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E						E		T

S

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

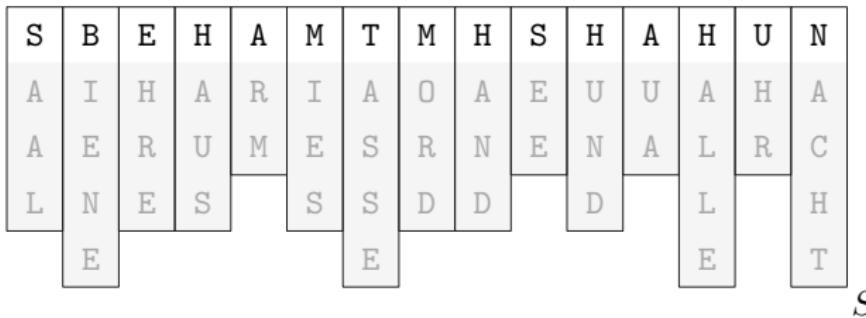
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

$$\begin{aligned} & \text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i), \\ & \text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1), \\ & \text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i) \end{aligned}$$


in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),

mkqSort($\langle e \in S : e[i] = p[i] \rangle, i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)

p

S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D	D		L		H	
	E					E					E			T

$i = 1$

S

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

B	E	A	A
I	H	R	U
E	R	M	A
N	E		
E			

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
			E			T

S

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

B	E	A	A
I	H	R	U
E	R	M	A
N	E		
E			

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
E			

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
E						T

S

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

B	E	A	A
I	H	R	U
E			
N	E		
E			

usW.

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
E			

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
E						T

S

in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

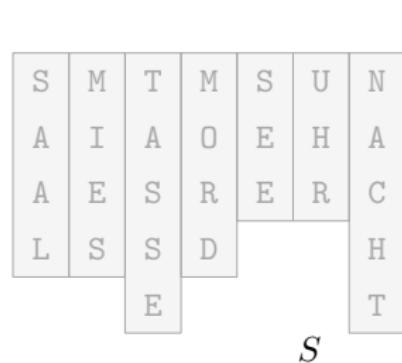
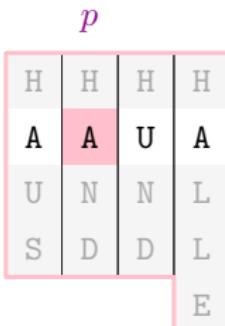
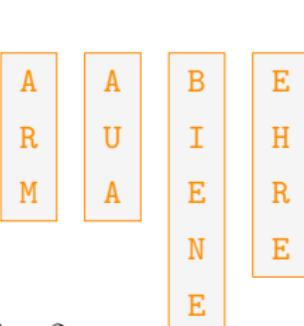
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

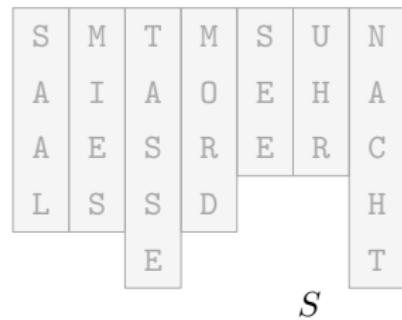
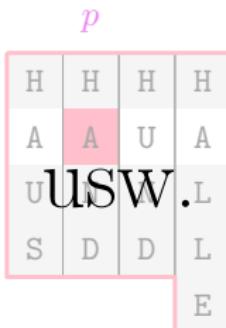
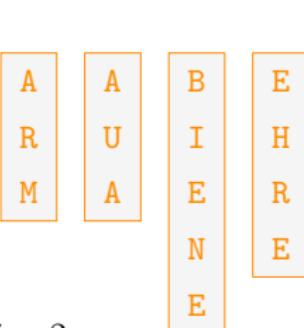
return concatenation of

(Rekursion)

mkqSort($\{e \in S : e[i] < p[i]\}$, i),

mkqSort($\{e \in S : e[i] = p[i]\}$, $i + 1$),

mkqSort($\{e \in S : e[i] > p[i]\}$, i)



in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

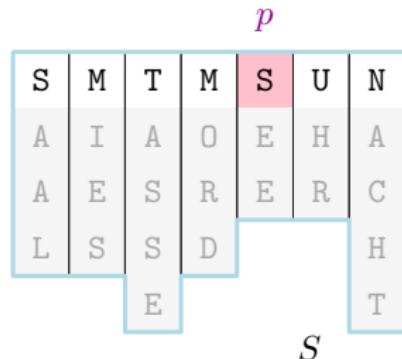
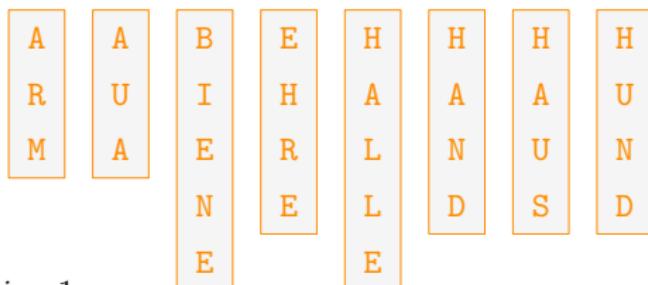
return concatenation of

(Rekursion)

mkqSort($\{e \in S : e[i] < p[i]\}$, i),

mkqSort($\{e \in S : e[i] = p[i]\}$, $i + 1$),

mkqSort($\{e \in S : e[i] > p[i]\}$, i)



in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

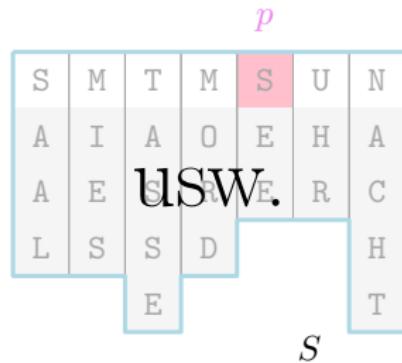
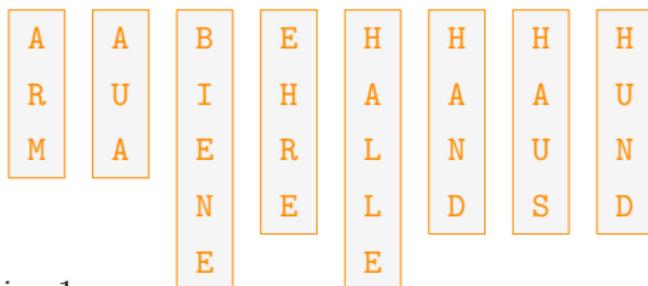
return concatenation of

(Rekursion)

mkqSort($\{e \in S : e[i] < p[i]\}$, i),

mkqSort($\{e \in S : e[i] = p[i]\}$, $i + 1$),

mkqSort($\{e \in S : e[i] > p[i]\}$, i)



in-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

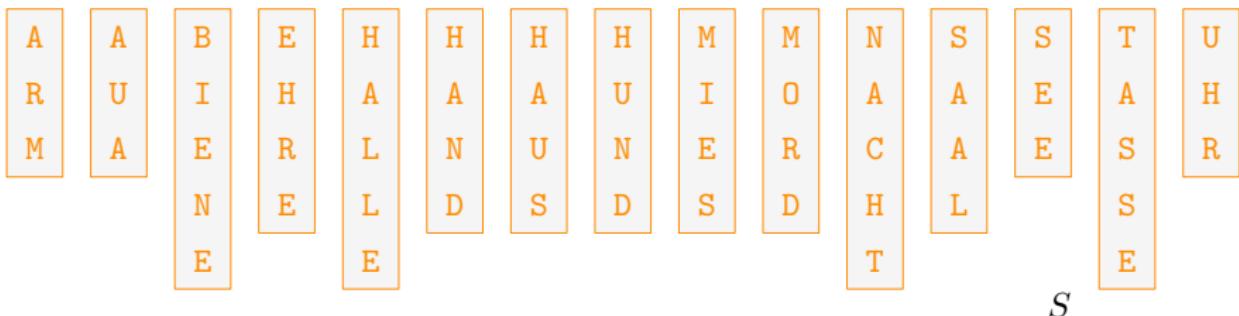
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

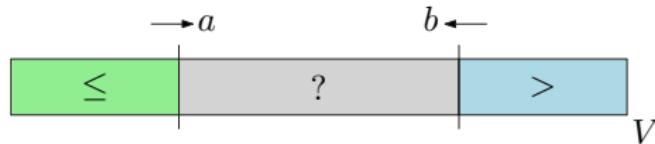


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

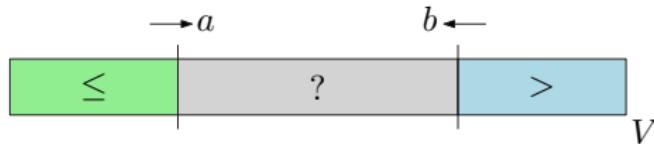


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p,$
 $b \rightarrow b - 1$, solange $V[b] > p,$
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$



in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p,$
 $b \rightarrow b - 1$, solange $V[b] > p,$
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

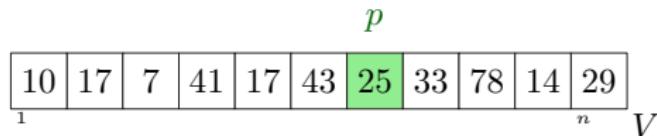
10	17	7	41	17	43	25	33	78	14	29
$_1$									$_n$	V

in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

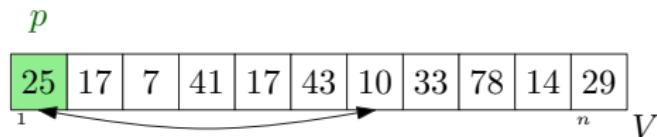


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

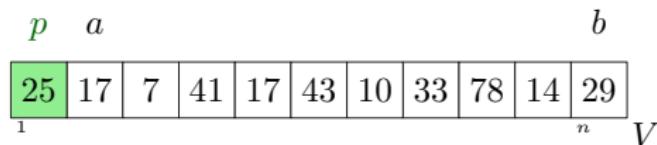


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p,$
 $b \rightarrow b - 1$, solange $V[b] > p,$
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

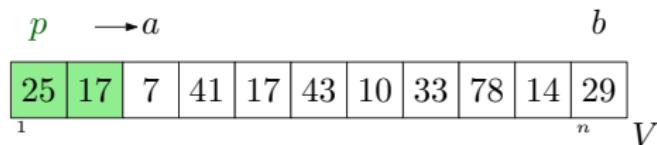


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p,$
 $b \rightarrow b - 1$, solange $V[b] > p,$
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

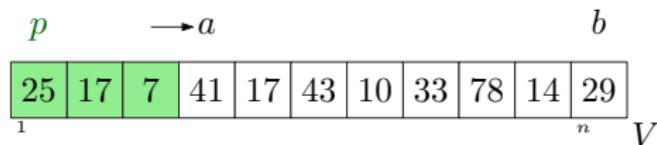


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

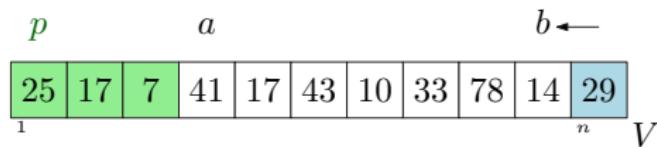


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

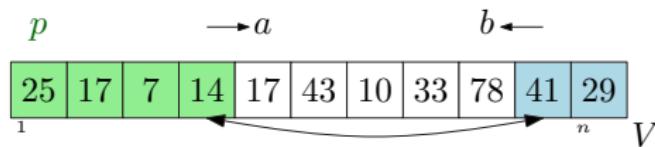


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

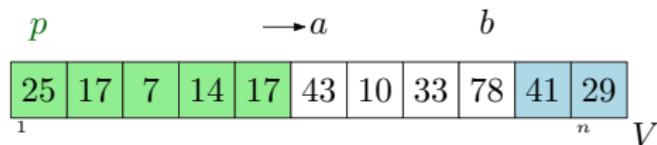


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p,$
 $b \rightarrow b - 1$, solange $V[b] > p,$
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

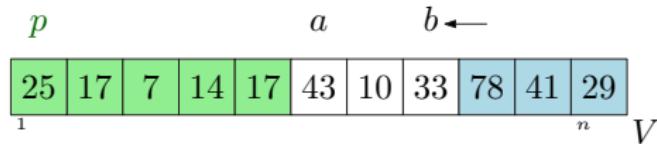


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p,$
 $b \rightarrow b - 1$, solange $V[b] > p,$
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

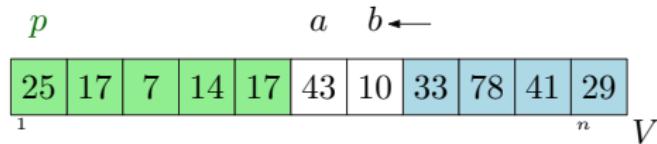


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

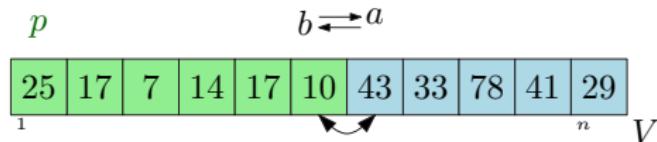


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
 - zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p, V[i > b] > p$
-
- Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

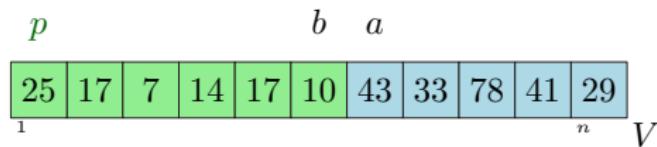


in-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
→ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
 - Wähle Pivot p und tausche mit erstem Element,
setze $a = 2, b = n$
 - $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
 - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
 - Ende, wenn $a > b$

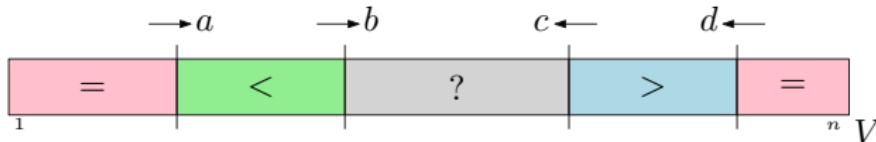


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
→ Invariante: $V[i \in [a, b] \text{ } a \neq b] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d) \text{ } c \neq d] > p$

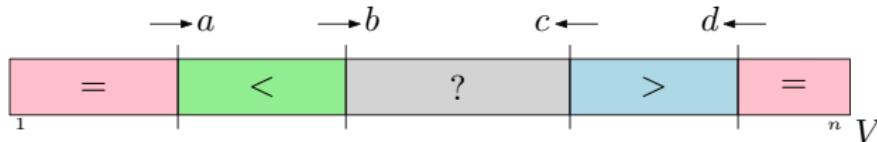


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
→ Invariante: $V[i \in [a, b) \ a \neq b] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d] \ c \neq d] > p$



in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
→ Invariante: $V[i \in [a, b) \ a \neq b] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d) \ c \neq d] > p$

1	S	B	E	H	A	M	T	M	H	S	H	A	H	U	N <th>n</th>	n
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A	
	A	E	R	U	M	E	S	R	N	E	N	A	L	R	C	
	L	N	E	S		S	S	D	D		D		L		H	
	E					E						E			T	

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
→ Invariante: $V[i \in [a, b) \ a \neq b] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d) \ c \neq d] > p$

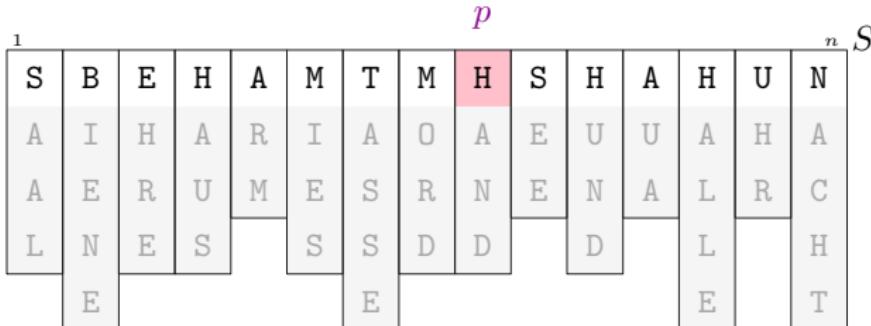
$$V \hat{=} \begin{array}{ccccccccccccccccccccc} & \\ & ^1 & S & B & E & H & A & M & T & M & H & S & H & A & H & U & N & & ^n & S \\ & & A & I & H & A & R & I & A & O & A & E & U & U & A & H & A & H & A & & & \\ & & A & E & R & U & M & E & S & R & N & E & N & A & L & R & C & & & \\ & & L & N & E & S & & S & S & D & D & & D & & L & & & & & \\ & & E & & & & & E & & & & & & & E & & & & & & \\ & \end{array}$$

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

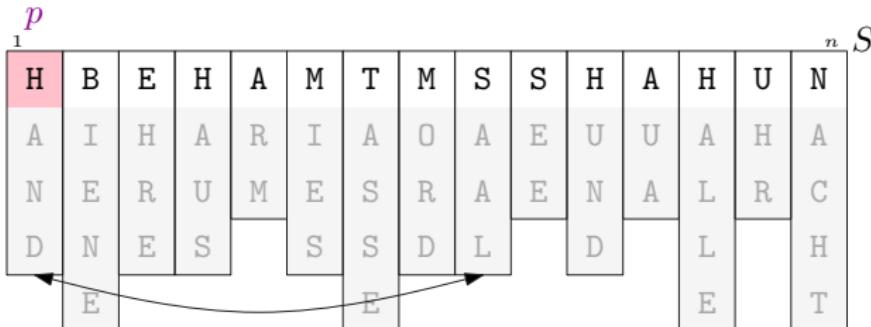


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

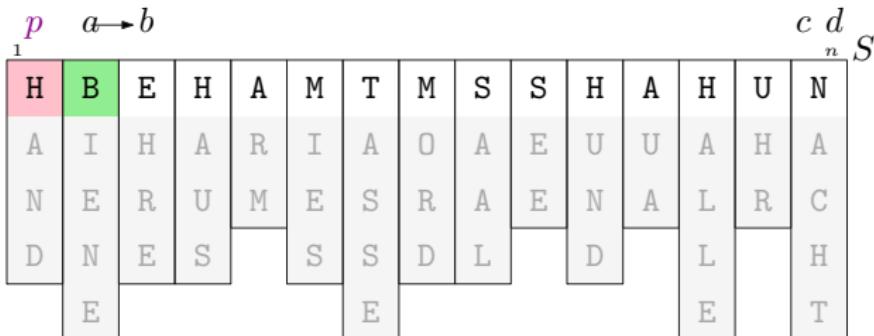
p	a	b	S													
1			c	d	n											
H	B	E	H	A	M	T	M	S	S	H	A	H	U	N		
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A		
N	E	R	U	M	E	S	R	A	E	N	A	L	R	C		
D	N	E	S		S	S	D	L		D		L	L	H		
E					E						E			T		

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

$\overset{p}{\textcolor{violet}{\texttt{H}}}$ $\overset{a}{\texttt{B}}$ $\overset{\longrightarrow b}{\texttt{E}}$

1	B	E	H	A	M	T	M	S	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
N	E	R	U	M	E	S	R	A	E	N	A	L	R	C
D	N	E	S		S	S	D	L		D		L	L	H
E					E						E			T

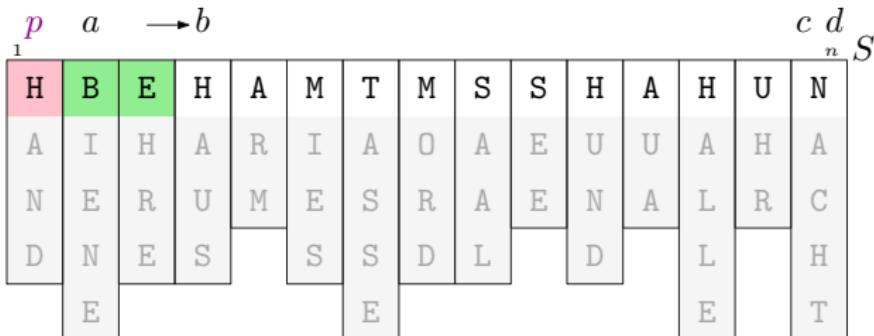
$\overset{c}{\texttt{S}}$ $\overset{d}{n}$

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

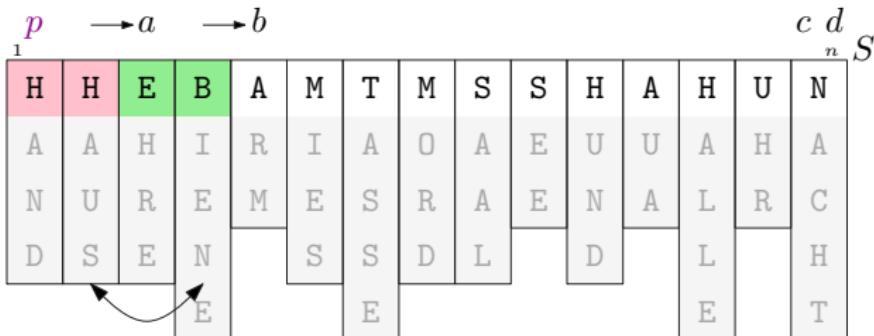


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

p	a	$\rightarrow b$				c	d	n	S
H	H	E	B	A	M	T	M	S	H
A	A	H	I	R	I	A	O	A	E
N	U	R	E	M	E	S	R	A	N
D	S	E	N		S	S	D	L	D
		E			E				L
									E
									T

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

p	a	b	$c \leftarrow d$		n	S
H	H	E	M	T	M	N
A	A	H	I	R	O	A
N	U	R	E	M	A	E
D	S	E	N	S	U	U
		E		E	A	A
					L	H
					R	C
					L	H
					E	T

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

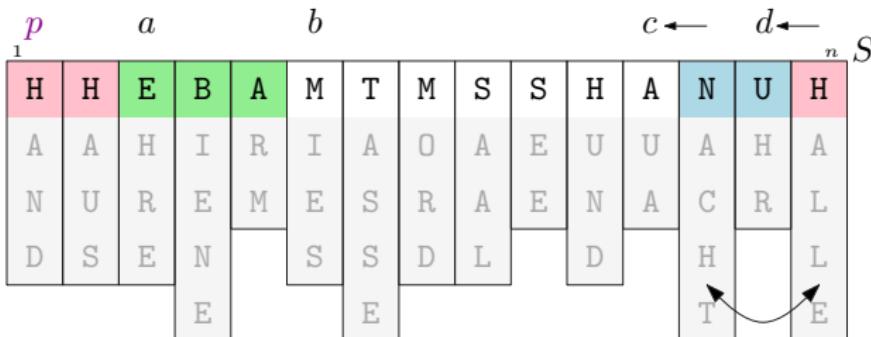
p	a	b			$c \leftarrow$	d	n	S
H	H	E	B	A	M	T	M	S
A	A	H	I	R	I	A	O	A
N	U	R	E	M	E	S	R	A
D	S	E	N		S	S	D	L
		E			E		D	
							L	
							E	
							T	

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

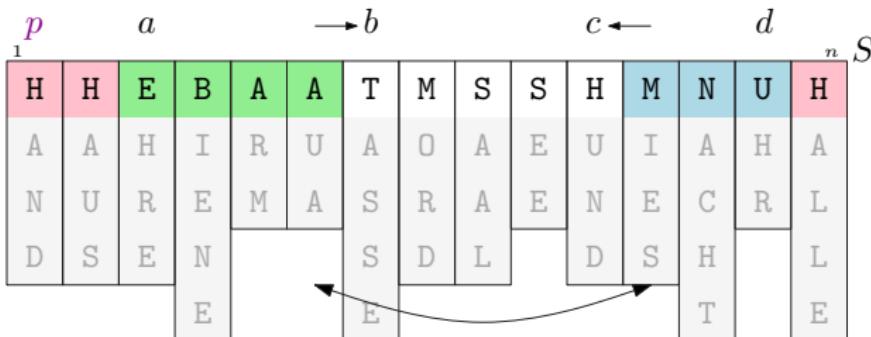


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

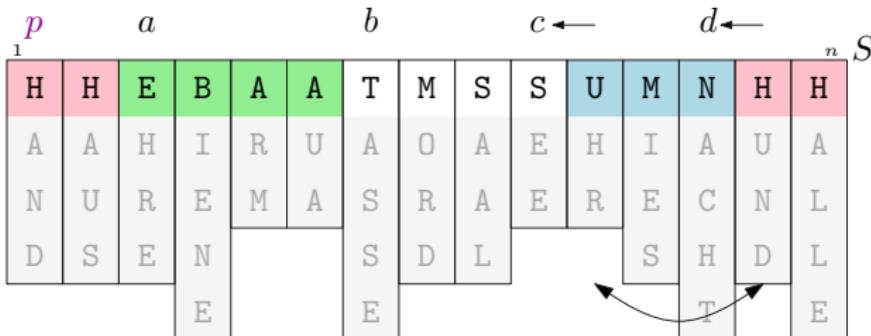


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

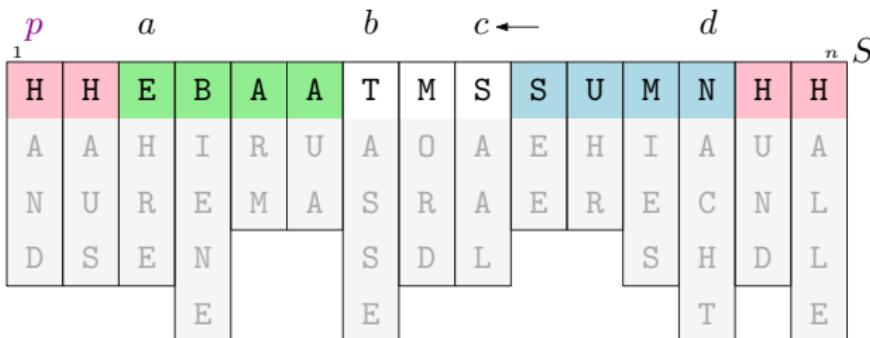


in-place Multikey Quicksort

Partitionierung

***in-place* bei Multikey Quicksort (Algorithmus)**

- Wähle Pivot p und tausche mit erstem Element,
setze $a = b = 2, c = d = n$
 - $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$,
 $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
 - Tausch, wenn $V[b] > p$ und $V[c] < p$
 - Ende, wenn $b > c$

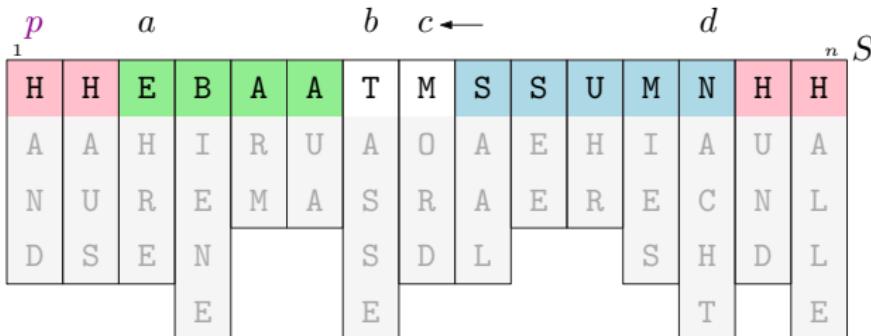


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$



in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

p	a	b	$c \leftarrow$	d	n	S
1	H H E B A A T M S S U M N H H					
	A A H I R U A O A E H I A U A					
	N U R E M A S R A E R E C N L					
	D S E N S D L S H D L L E					
	E					

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

p	a	$c \leftarrow b$	d	n
1				S
H H E B A A T M S S U M N H H				
A A H I R U A O A E H I A U A				
N U R E M A S R A E R E C N L				
D S E N S D L S H D L L E				
E				

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a - 1, b - a)$
Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$

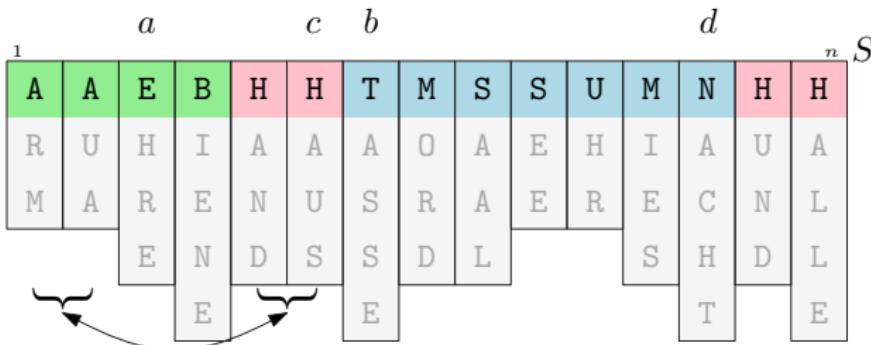
1	a		c		b		d		n		S			
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
D	S	E	N			S	D	L			S	H	D	L
		E				E					T		E	

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a - 1, b - a)$
Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$



in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a - 1, b - a)$
Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$

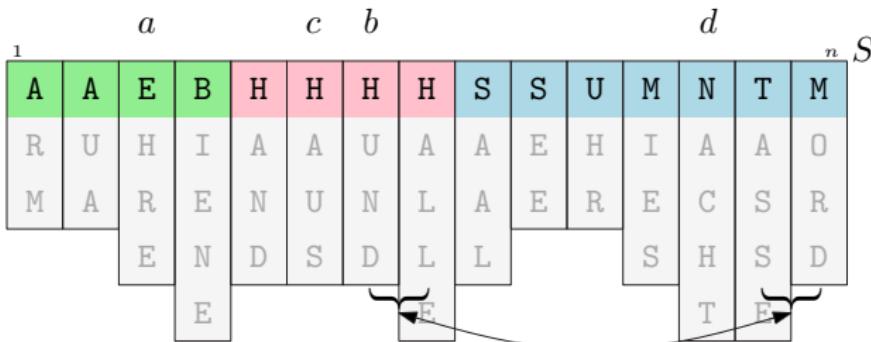
a	c	b	d	n
A	H	T	M	S
R	I	O	I	A
M	A	A	E	U
A	N	S	C	A
R	U	R	N	L
E	S	A	H	L
E	D	E	S	L
	S	D	H	E
		L	T	

in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a - 1, b - a)$
Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$

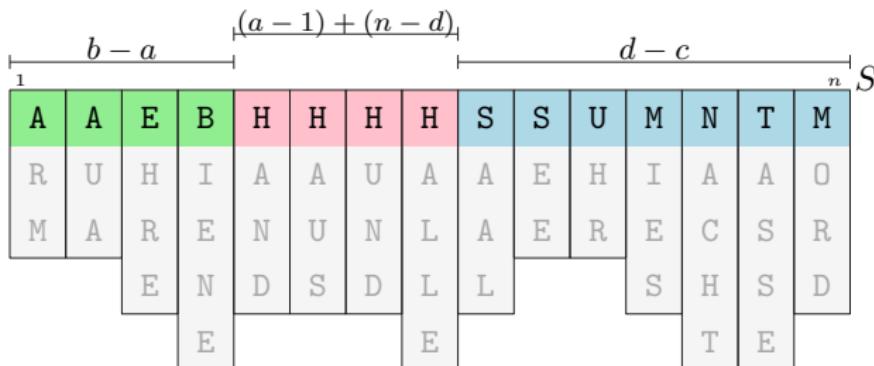


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a - 1, b - a)$
Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$



in-place Multikey Quicksort

Zusammenfassung

- *Three-way Radix Quicksort*

(Partitionierung in kleiner, gleich, größer über alle Stellen analog zu msd-Radixsort)

- effizient $\mathcal{O}(|S| \log |S| + d)$

($d \triangleq$ Summe der Länge der unterscheidenden Präfixe)

- *in-place* Partitionierung möglich

(durch geschicktes Speichern und Verschieben der gleichen Elemente)

- sehr einfache Implementierung

(nur 40 Zeilen Quellcode, siehe Anhang)

Suche mit Suffix-Arrays

Wiederholung

gesucht

- Anzahl / Position aller Vorkommen von Muster P in Text T
(counting query / reporting query)

Ansatz

- Suffix-Array SA von T indiziert alle Suffixe in sortierter Reihenfolge
 - **binäre Suche** in SA für erstes, letztes Vorkommen von P in T
 - über Indizes s, t alle Vorkommen bestimmt

Laufzeit

- $\mathcal{O}(m \log n)$ bzw. $\mathcal{O}(m \log n + occ(p))$
(mit $|P| = m, |T| = n, occ(p) = \text{Anzahl Vorkommen von } P \text{ in } T$)

Suche mit Suffix-Arrays

Ablauf

$T = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ b & a & r & b & a & r & h & a & b & a & r & b & e & r & \$ \end{matrix}$

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

■ (SA bestimmen)

■ finde Start

■ finde Ende

■ Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	T = b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i															
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
2		a	r	b	a	r	h	a	b	a	r	b	e	r	\$
3		r	b	a	r	h	a	b	a	r	b	e	r	\$	
4			b	a	r	h	a	b	a	r	b	e	r	\$	
5				a	r	h	a	b	a	r	b	e	r	\$	
6					r	h	a	b	a	r	b	e	r	\$	
7						h	a	b	a	r	b	e	r	\$	
8							a	b	a	r	b	e	r	\$	
9								b	a	r	b	e	r	\$	
10									a	r	b	e	r	\$	
11										r	b	e	r	\$	
12										b	e	r	\$		
13										e	r	\$			
14										r	\$				
15											\$				

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

■ (SA bestimmen)

■ finde Start

■ finde Ende

■ Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i SA[i]															
1 15 \$															
2 8 a b a r b e r \$															
3 2 a r b a r h a b a r b e r \$															
4 10 a r b e r \$															
5 5 a r h a b a r b e r \$															
6 1 b a r b a r h a b a r b e r \$															
7 9 b a r b e r \$															
8 4 b a r h a b a r b e r \$															
9 12 b e r \$															
10 13 e r \$															
11 7 h a b a r b e r \$															
12 14 r \$															
13 3 r b a r h a b a r b e r \$															
14 11 r b e r \$															
15 6 r h a b a r b e r \$															

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start (binäre Suche)

$I = 1, r = n$

while ($I < r$) **do**

$$q = \lfloor \frac{I+r}{2} \rfloor$$

if ($P > T_{SA[q]..SA[q]+m-1}$)

then $I = q + 1$

else $r = q$

$s = I$

if ($P \neq T_{SA[s]..SA[s]+m-1}$)

then break

- finde Ende

- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15														
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
9	12	b	e	r	\$										
10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$l = 1, r = 16$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start (binäre Suche)

$I = 1, r = n$

while ($I < r$) **do**

$q = \lfloor \frac{l+r}{2} \rfloor$

if ($P > T_{SA[q]..SA[q]+m-1}$)

then $I = q + 1$

else $r = q$

$s = I$

if ($P \neq T_{SA[s]..SA[s]+m-1}$)

then break

- finde Ende

- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15														
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
9	12	b	e	r	\$										
10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$l = 1, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start (binäre Suche)

$I = 1, r = n$

while ($I < r$) **do**

$$q = \lfloor \frac{l+r}{2} \rfloor$$

if ($P > T_{SA[q]..SA[q]+m-1}$) $q =$

then $I = q + 1$

else $r = q$

$s = I$

if ($P \neq T_{SA[s]..SA[s]+m-1}$) $s =$

then break

- finde Ende

- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i $\text{SA}[i]$															
1 15 \$															
2 8 a b a r b e r \$															
3 2 a r b a r h a b a r b e r \$															
4 10 a r b e r \$															
5 5 a r h a b a r b e r \$															
6 1 b a r b a r h a b a r b e r \$															
7 9 b a r b e r \$															
8 4 b a r h a b a r b e r \$															
9 12 b e r \$															
10 13 e r \$															
11 7 h a b a r b e r \$															
12 14 r \$															
13 3 r b a r h a b a r b e r \$															
14 11 r b e r \$															
15 6 r h a b a r b e r \$															

$l = 5, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start (binäre Suche)
 $I = 1, r = n$
- while ($I < r$) do
 - $q = \lfloor \frac{l+r}{2} \rfloor$
 - if ($P > T_{SA[q]..SA[q]+m-1}$)
 - then $I = q + 1$
 - else $r = q$
 - $s = I$
 - if ($P \neq T_{SA[s]..SA[s]+m-1}$)
 - then break
 - finde Ende
 - Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	$T = b$	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15	\$													
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
9	12	b	e	r	\$										
10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$$l = 5, r = 6$$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start (binäre Suche)
 $I = 1, r = n$
while ($I < r$) **do**
 $q = \lfloor \frac{l+r}{2} \rfloor$
if ($P > T_{SA[q]..SA[q]+m-1}$)
 then $I = q + 1$
 else $r = q$
 $s = I$
if ($P \neq T_{SA[s]..SA[s]+m-1}$)
 then break
- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i SA[i]															
1 15 \$															
2 8 a b a r b e r \$															
3 2 a r b a r h a b a r b e r \$															
4 10 a r b e r \$															
5 5 a r h a b a r b e r \$															
6 1 b a r b a r h a b a r b e r \$															
7 9 b a r b e r \$															
8 4 b a r h a b a r b e r \$															
9 12 b e r \$															
10 13 e r \$															
11 7 h a b a r b e r \$															
12 14 r \$															
13 3 r b a r h a b a r b e r \$															
14 11 r b e r \$															
15 6 r h a b a r b e r \$															

$$l = 6, r = 6$$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start
- finde Ende (binäre Suche)

$l = s, r = n$

while ($l < r$) **do**

$q = \lceil \frac{l+r}{2} \rceil$

if ($P = T_{SA[q]..SA[q]+m-1}$)

then $l = q$

else $r = q - 1$

$t = l$

- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15	\$													
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
9	12	b	e	r	\$										
q=10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$l = 5, r = 15$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

■ (SA bestimmen)

■ finde Start

■ finde Ende (binäre Suche)

$I = s, r = n$

while ($I < r$) **do**

$q = \lceil \frac{I+r}{2} \rceil$

if ($P = T_{SA[q]..SA[q]+m-1}$)

then $I = q$

else $r = q - 1$

$t = I$

■ Ergebnis

$T = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ b & a & r & b & a & r & h & a & b & a & r & b & e & r & \$ \end{matrix}$

i $SA[i]$

1 15 \$

2 8 a b a r b e r \$

3 2 a r b a r h a b a r b e r \$

4 10 a r b e r \$

5 5 a r h a b a r b e r \$

6 1 b a r b a r h a b a r b e r \$

7 9 b a r b e r \$

8 4 b a r h a b a r b e r \$

9 12 b e r \$

10 13 e r \$

11 7 h a b a r b e r \$

12 14 r \$

13 3 r b a r h a b a r b e r \$

14 11 r b e r \$

15 6 r h a b a r b e r \$

$l = 5, r = 9$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

■ (SA bestimmen)

■ finde Start

■ finde Ende (binäre Suche)

$I = s, r = n$

while ($I < r$) **do**

$q = \lceil \frac{I+r}{2} \rceil$

if ($P = T_{SA[q]..SA[q]+m-1}$) $q =$

then $I = q$

else $r = q - 1$

$t = I$

■ Ergebnis

$T = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ b & a & r & b & a & r & h & a & b & a & r & b & e & r & \$ \end{matrix}$

i $SA[i]$

1 15 \$

2 8 a b a r b e r \$

3 2 a r b a r h a b a r b e r \$

4 10 a r b e r \$

5 5 a r h a b a r b e r \$

6 1 b a r b a r h a b a r b e r \$

7 9 b a r b e r \$

8 4 b a r h a b a r b e r \$

9 12 b e r \$

10 13 e r \$

11 7 h a b a r b e r \$

12 14 r \$

13 3 r b a r h a b a r b e r \$

14 11 r b e r \$

15 6 r h a b a r b e r \$

$l = 7, r = 9$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

■ (SA bestimmen)

■ finde Start

■ finde Ende (binäre Suche)

$I = s, r = n$

while ($I < r$) **do**

$q = \lceil \frac{I+r}{2} \rceil$

if ($P = T_{SA[q]..SA[q]+m-1}$)

then $I = q$

else $r = q - 1$

$t = I$

■ Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15	\$													
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
q = 9	12	b	e	r	\$										
10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$$l = 8, r = 9$$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start
- finde Ende (binäre Suche)

$l = s, r = n$

while ($l < r$) **do**

$$q = \lceil \frac{l+r}{2} \rceil$$

if ($P = T_{SA[q]..SA[q]+m-1}$)

then $l = q$

else $r = q - 1$

$t = l$

- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15	\$													
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
9	12	b	e	r	\$										
10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$$l = 8, r = 8$$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis
 - $t - s + 1$
(counting query)
 - $\{SA[s], \dots, SA[t]\}$
(reporting query)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	SA[i]														
1	15	\$													
2	8	a	b	a	r	b	e	r	\$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
4	10	a	r	b	e	r	\$								
5	5	a	r	h	a	b	a	r	b	e	r	\$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	\$
7	9	b	a	r	b	e	r	\$							
q = 8	4	b	a	r	h	a	b	a	r	b	e	r	\$		
9	12	b	e	r	\$										
10	13	e	r	\$											
11	7	h	a	b	a	r	b	e	r	\$					
12	14	r	\$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$	
14	11	r	b	e	r	\$									
15	6	r	h	a	b	a	r	b	e	r	\$				

$$s = 6, t = 8$$

Suche mit Suffix-Arrays

Zusammenfassung

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
 - einmal Suffix-Array generieren in $\mathcal{O}(n)$,
 - danach **Anfragen in $\mathcal{O}(m \log n)$** möglich, statt in $\mathcal{O}(mn)$
(beides auch in $\mathcal{O}(m + \log n)$ bzw. $\mathcal{O}(m + n)$ möglich)
(gut, wenn auf einem Text viele Anfragen stattfinden)
- Ausnutzung der Eigenschaften des Suffix-Arrays
 - jeder Substring ist Präfix eines Suffix
 - **alle Substrings liegen "sortiert" vor**
(mögliche Ausnahme: Substring ist Präfix von Substring)
(das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge)

Definition:

- LCP[i]: Länge des längsten gemeinsamen Präfixes von je zwei lexikographisch benachbarten Suffixen $A[SA[i-1] \dots n]$ und $A[SA[i] \dots n]$

Erweiterung auf beliebige Suffixe

- LCP[i][j]: Länge des längsten gemeinsamen Präfix **beliebiger lexikographischer** Suffixe $A[SA[i] \dots n]$ und $A[SA[j] \dots n]$
- Konstruktion: $\mathcal{O}(n \log n)$ Zeit und Platz
- Zugriff: $\mathcal{O}(1)$

Schnelle Suche mit Suffix-Arrays

Erster Ansatz

Suche: $P = \text{bar}$

- Ziel: kein wiederholtes Vergleichen von Zeichen aus P
- Nutze LCP-Array um Suche zu beschleunigen
- Starte Suche bei mlr
 - $I := \text{LCP}[L][P]$
 - $r := \text{LCP}[R][P]$
 - $mlr := \min(I, r)$
 - Update von I, r , keine Neuberechnung
- Oft $\mathcal{O}(m + \log n)$
- Worst case $\mathcal{O}(m \log n)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$	
i SA[i]																
1	15														\$	
2	8	a	b	a	r	b	e	r							\$	
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$	
4	10	a	r	b	e	r									\$	
5	5	a	r	h	a	b	a	r	b	e	r				\$	
$L=6$	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
$q=7$	9	b	a	r	b	e	r									
8	4	b	a	r	h	a	b	a	r	b	e	r				\$
$R=9$	12	b	e	r												
10	13	e	r													
11	7	h	a	b	a	r	b	e	r							
12	14	r														
13	3	r	b	a	r	h	a	b	a	r	b	e	r		\$	
14	11	r	b	e	r											
15	6	r	h	a	b	a	r	b	e	r						

$$L = 6, R = 9$$

$$l = 3, r = 1$$

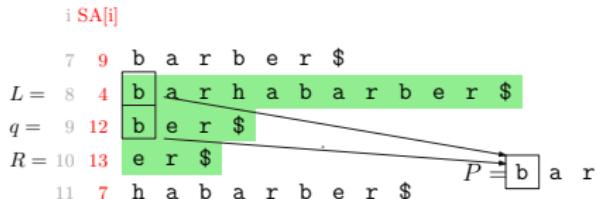
$$mlr := 1 = \min(l, r)$$

Schnelle Suche mit Suffix-Arrays

Redundante Vergleiche

Problem

- Falls $l \neq r \rightarrow$ wiederholtes Vergleichen



Definition

- Vergleich eines Zeichens aus P ist **redundant**, falls das Zeichen vorher schon einmal überprüft wurde.

Ziel

- Beschränke redundante Vergleiche auf $\mathcal{O}(1)$ pro Iteration
- Vergleiche bei $\max(l, r)$ beginnen

Suche mit Suffix-Arrays

Ablauf

Ansatz

- **if** ($I = r$)
start at mlr
Update I, r, L, R
- **if** ($I > r \wedge \text{LCP}[L, q] > I$)
 $L := q + 1$
Update I
- **if** ($I > r \wedge \text{LCP}[L, q] < I$)
 $R := q$
 $r := \text{LCP}[L, q]$
- **if** ($I > r \wedge \text{LCP}[L, q] = I$)
start at I

Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P = \text{barberac}$

- **if** ($I = r$)

start at m/r

Update I, r, L, R

- **if** ($I > r \wedge \text{LCP}[L, q] > I$)

b a r b e r a b a ...

- **if** ($I > r \wedge \text{LCP}[L, q] < I$)

b a r b e r a b c ...

- **if** ($I > r \wedge \text{LCP}[L, q] = I$)

b a r b e r a c c ...

b a r b e r c b c \$

b a r b i

$l = 4, r = 4$

Suche mit Suffix-Arrays

Ablauf

$L = \boxed{\text{b a r b}} \text{ a r h a b a r b e r ...}$

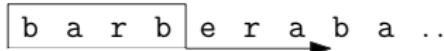
Suche: $P = \text{barberac}$

- **if** ($I = r$)

start at m/r

Update I, r, L, R

- **if** ($I > r \wedge \text{LCP}[L, q] > I$) $q =$

 $\boxed{\text{b a r b}} \text{ e r a b a ...}$

- **if** ($I > r \wedge \text{LCP}[L, q] < I$)

$\boxed{\text{b a r b}} \text{ e r a b b ...}$

- **if** ($I > r \wedge \text{LCP}[L, q] = I$)

$\begin{array}{c} \text{b a r b e r a b c ...} \\ \text{b a r b e r a c c \$} \\ \text{b a r b e r c b c ...} \end{array} \quad \text{LCP}[L, q] = 4$

$R = \boxed{\text{b a r b i}}$ $l = 4, r = 4$

Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P = \text{barberac}$

- **if** ($l = r$)
- **if** ($l > r \wedge \text{LCP}[L, q] > l$)
- **if** ($l > r \wedge \text{LCP}[L, q] < l$)
- **if** ($l > r \wedge \text{LCP}[L, q] = l$) $L =$

b a r b e r a b a ...
b a r b e r a b b ...

b a r b e r a b c ...
b a r b e r a c c \\$
b a r b e r c b c ...
 $R =$ b a r b i $l = 7, r = 4$

Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P = \text{barberac}$

- **if** ($I = r$)
- **if** ($I > r \wedge \text{LCP}[L, q] > I$)

$$L := q + 1$$

Update I

- **if** ($I > r \wedge \text{LCP}[L, q] < I$) $L =$
- **if** ($I > r \wedge \text{LCP}[L, q] = I$)

b a r b e r a b a ...

b a r b e r a b b ...

$$q =$$

b a r b e r a b c ...

b a r b e r a c c \$

b a r b e r c b c ... $\text{LCP}[L, q] = 8$

$$R =$$

b a r b i

$$l = 7, r = 4$$

Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P = \text{barberac}$

- **if** ($I = r$)
- **if** ($I > r \wedge \text{LCP}[L, q] > I$)
- **if** ($I > r \wedge \text{LCP}[L, q] < I$)

$R := q$

$r := \text{LCP}[L, q]$

b a r b e r a b a ...

b a r b e r a b b ...

- **if** ($I > r \wedge \text{LCP}[L, q] = I$)

$L =$ b a r b e r a b c ...
 $q =$ b a r b e r a c c \$
 $R =$ b a r b i

$\text{LCP}[L, q] = 6$
 $l = 8, r = 4$

Suche mit Suffix-Arrays

Ablauf

b	a	r	b	a	r	h	a	b	a	r	b	e	r	...
---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

Suche: $P = \text{barberac}$

- **if** ($I = r$)
- **if** ($I > r \wedge \text{LCP}[L, q] > I$)
- **if** ($I > r \wedge \text{LCP}[L, q] < I$)

$R := q$

$r := \text{LCP}[L, q]$

b a r b e r a b a ...

b a r b e r a b b ...

- **if** ($I > r \wedge \text{LCP}[L, q] = I$)

b a r b e r a b c ...

b a r b e r a c c \$

b a r b e r c b c ...

b a r b i

$l = 8, r = 6$

Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P = \text{barberac}$

- **if** ($l = r$)
- **if** ($l > r \wedge \text{LCP}[L, q] > l$)
- **if** ($l > r \wedge \text{LCP}[L, q] < l$)
- **if** ($l > r \wedge \text{LCP}[L, q] = l$)

b a r b e r a b a ...

b a r b e r a b b ...

$L = q =$ b a r b e r a b c ...
 $R =$ b a r b e r a c c \$
b a r b e r c b c ...
b a r b i

$\text{LCP}[L, q] = 10$
 $l = 8, r = 6$

Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P = \text{barberac}$

- **if** ($I = r$)
 - **if** ($I > r \wedge \text{LCP}[L, q] > I$)
 - **if** ($I > r \wedge \text{LCP}[L, q] < I$)
 - **if** ($I > r \wedge \text{LCP}[L, q] = I$)
start at I
- b a r b e r a b a ...
b a r b e r a b b ...

b a r b e r a b c ...
 $L = R = \boxed{\text{b a r b e r a c}} \text{ c } \$$
b a r b e r c b c ...
b a r b i

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{barberac}$

- **if** ($I = r$)
- **if** ($I > r \wedge \text{LCP}[L, q] > I$)
- **if** ($I > r \wedge \text{LCP}[L, q] < I$)
- **if** ($I > r \wedge \text{LCP}[L, q] = I$)

Laufzeit

- LCP + SA: $\mathcal{O}(m + \log n)$ Vergleiche
- Beweisidee
 - I, r werden nur größer
 - Anzahl an redundanten Vergleichen pro Rekursion konstant

Suche mit Suffix-Arrays

Zusammenfassung

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
 - einmal Suffix-Array generieren in $\mathcal{O}(n)$,
 - danach **Anfragen in $\mathcal{O}(m \log n)$** möglich, statt in $\mathcal{O}(m + n)$
(gut, wenn auf einem Text viele Anfragen stattfinden)
- Verhindern redundanter Vergleiche
 - einmal Suffix-Array generieren in $\mathcal{O}(n)$,
 - einmal LCP-Array generieren in $\mathcal{O}(n)$,
 - einmal erweitertes LCP-Array generieren in $\mathcal{O}(n \log n)$,
 - danach **Anfrage in $\mathcal{O}(m + \log n)$**
- Ausnutzung der Eigenschaften des Suffix-Arrays
 - jeder Substring ist Präfix eines Suffix
 - **alle Substrings liegen "sortiert" vor**
(mögliche Ausnahme: Substring ist Präfix von Substring)
(das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge)



Feierabend!

in-place Multikey Quicksort

Definitionen

- nach C-Standard ist ein string ist Zeiger auf char
(die Daten werden als char Array mit Sentinel '\0' gespeichert)

```
void swap(int a, int b, char *x[]){
    char *t=x[a];
    x[a]=x[b];
    x[b]=t;
}
```

```
#define i2c(i) x[i][depth] //Buchstabe an Stelle 'depth' aus i-tem String
```

```
void vecswap(int i, int j, int n, char *x[]) { //Tausch der Sub-Arrays [i:i+n] und [j:j+n]
    while(n--> 0) {
        swap(i,j);
        i++; j++;
    }
}
```

in-place Multikey Quicksort Algorithmus

```
void mkqsort( char *x[], int n, int depth){
    if(n <= 1) return;
    a = rand() % n;
    swap(0,a,x);
    v=i2c(0);
    a = b = 1;
    c = d = n-1;
    while(true) {
        while(b <= c && (r = i2c(b) -v) <= 0) {
            if(r == 0) { swap(a,b,x); a++ }
            b++;
        }
        while (b <= c && (r= i2c(c) -v) >= 0) {
            if(r == 0) { swap(c,d,x); d--; }
            c--;
        }
        if(b>c) break;
        swap(b,c,x); b++; c--;
    }
    r = min(a, b-a); vecswap(0, b-r, r, x);
    r = min(d-c, n-d-1); vecswap(b, n-r, r, x);
    r = b-a; mkqsort(x, r, depth);
    if(i2c(r) != 0) { mkqsort(x+r, a+n-d-1, depth+1); } // "mittlere" Rekursion, falls notwendig
    r = d-c; mkqsort(x+n-r, r, depth);
}
```

// Weniger als zwei Strings zu sortieren?
// Wahl des Pivot-Elements
// Pivot an den Anfang des Arrays verschieben
// Wert der aktuellen Stelle im Pivot
// Anfang des zu sortierenden (Sub-)Arrays
// Ende des zu sortierenden (Sub-)Arrays

// Von links Elemente scannen, bis Char > Pivot
// Char == Pivot? Dann nach links kopieren
// Grenze der gleichen Elemente eins nach rechts

// Von rechts Elemente scannen, bis Char < Pivot
// Char == Pivot? Dann nach rechts kopieren
// Grenze der gleichen Elemente eins nach links

// Zeiger aneinander vorbei? Dann Abbruch
// Inversion gefunden, tauschen, Zeiger je 1 weiter

// "gleiche" Elemente von links zur Mitte kopieren
// "gleiche" Elemente von rechts zur Mitte kopieren
// "linke" Rekursion
// "rechte" Rekursion

[Bentley&Sedgewick1997]

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

$$\begin{aligned} & \text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i), \\ & \text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1), \\ & \text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i) \end{aligned}$$

S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E						E		T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

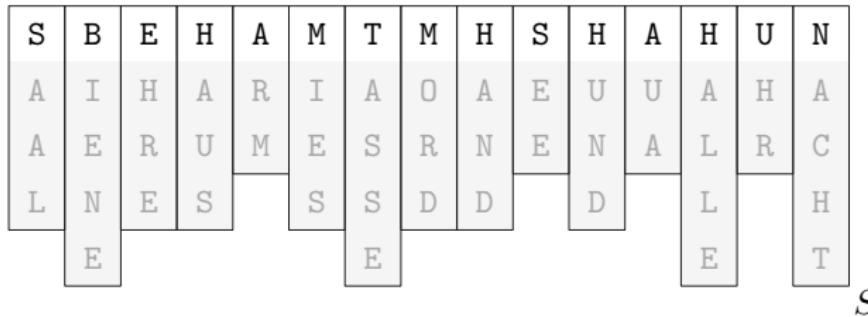
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

$$\begin{aligned} & \text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i), \\ & \text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1), \\ & \text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i) \end{aligned}$$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$

p

S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E					E			T

$i = 1$

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

B	E	A	A
I	H	R	U
E	R	M	A
N	E		
E			

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
			E			T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

B	E	A	A
I	H	R	U
E	R	M	A
N	E		
E			

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
E						T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

A	A
R	U
M	A

$i = 1$

B
I
H
E
R
N
E

E
H
A
U
A
N
N
L
S
D
D
L
E

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
			E			T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

A	A
R	U
M	A

$i = 1$

B
I
H
E
R
N
E

E
H
R
N
E

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
			E			T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

A	A
R	U
M	A

$i = 1$

B
I
H
E
N
E

E
H
R
E
E

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
			E			T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p

A	A
R	U
M	A

$i = 2$

B
I
H
E
N
E

E
H
R
E

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E	R	C
L	S	S	D			H
			E			T

S

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

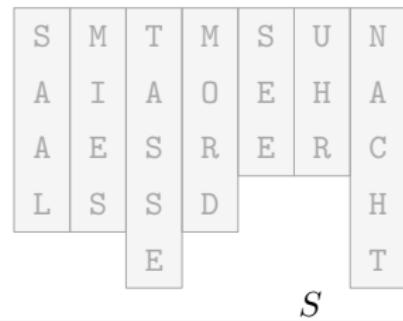
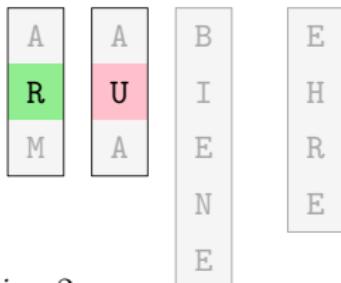
(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

p



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

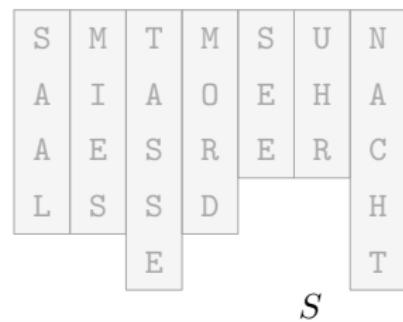
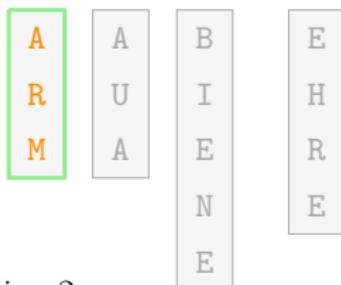
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

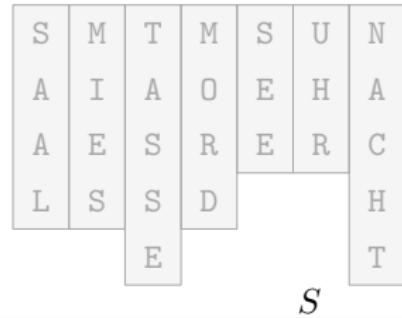
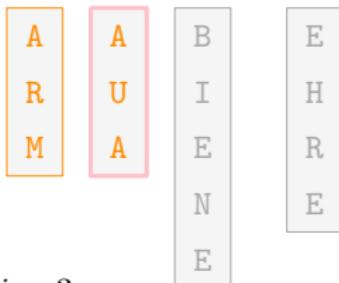
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

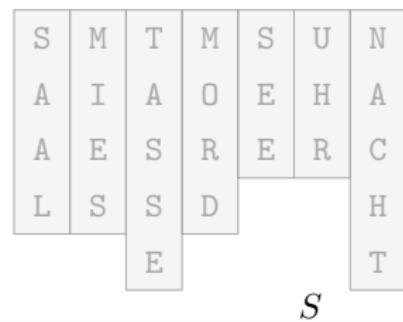
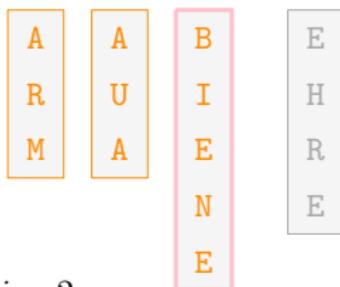
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

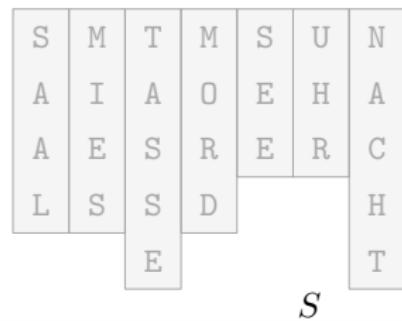
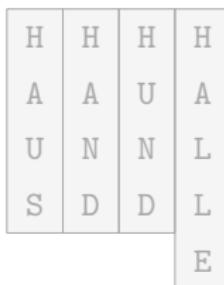
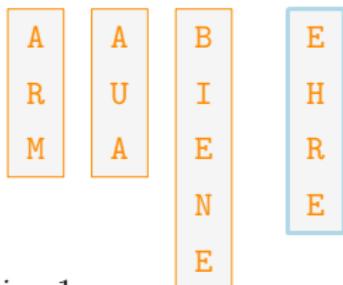
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

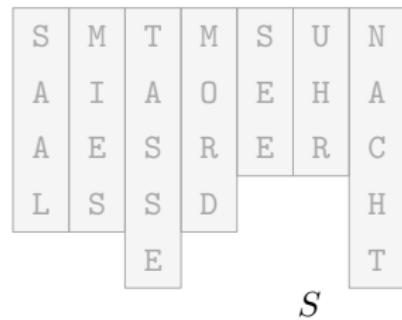
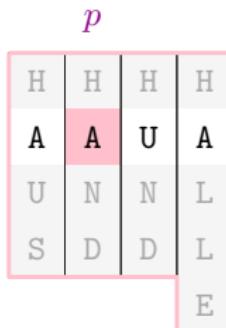
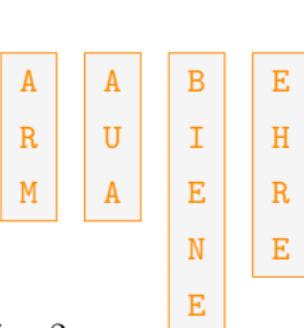
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

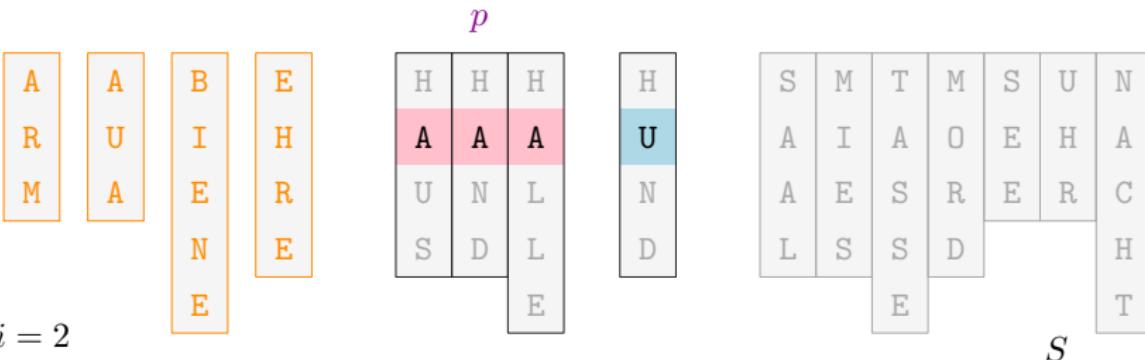
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

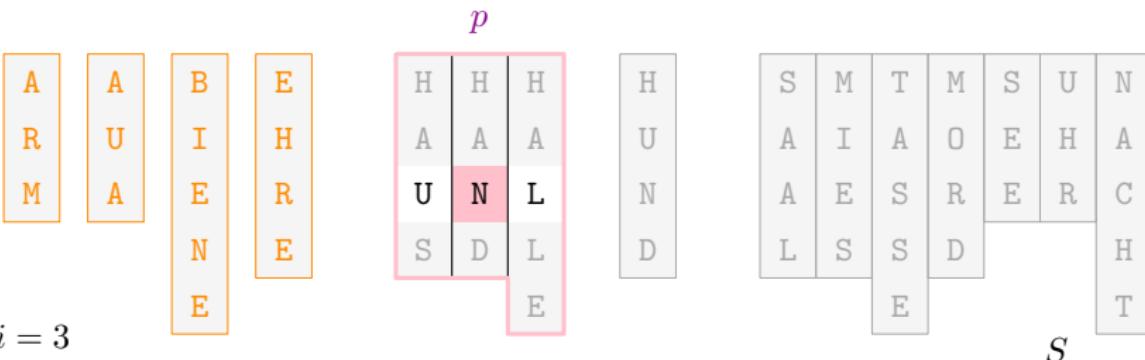
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

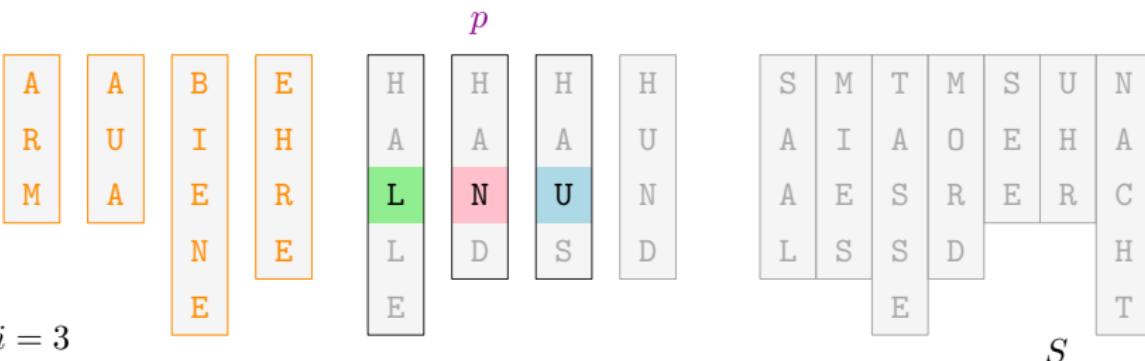
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

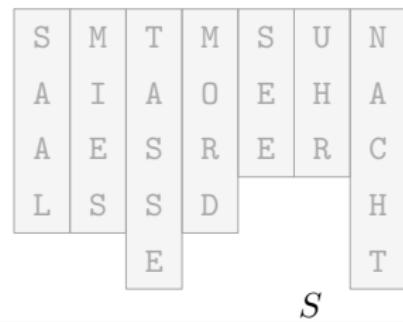
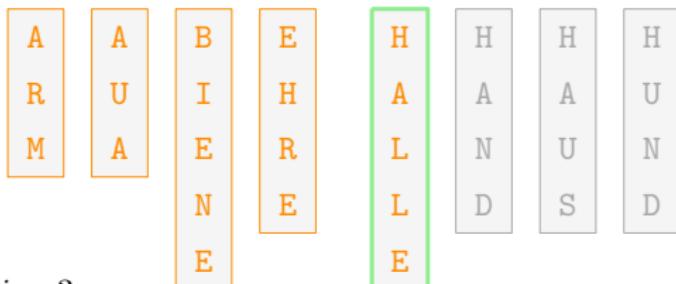
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

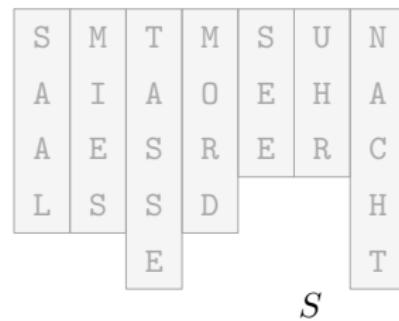
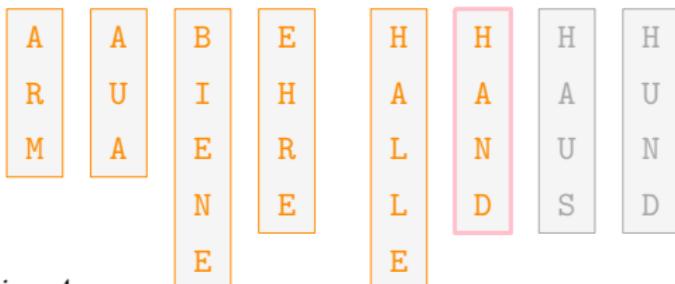
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

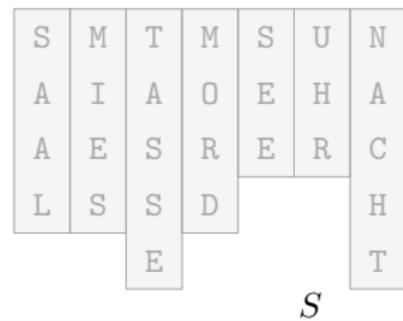
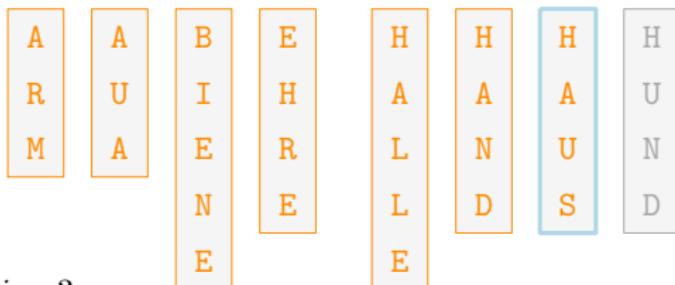
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

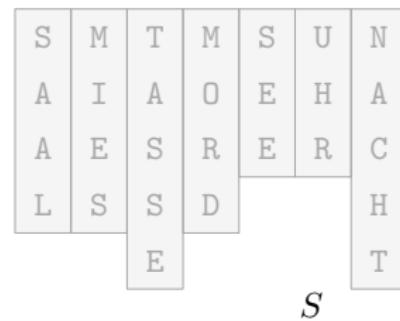
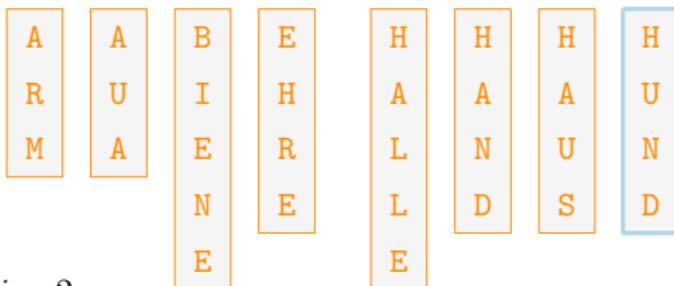
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

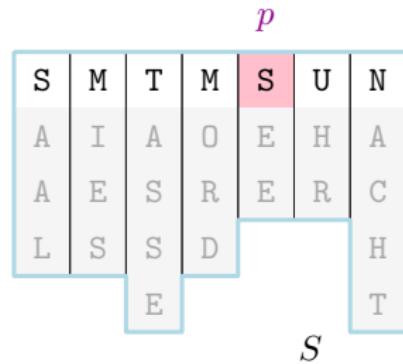
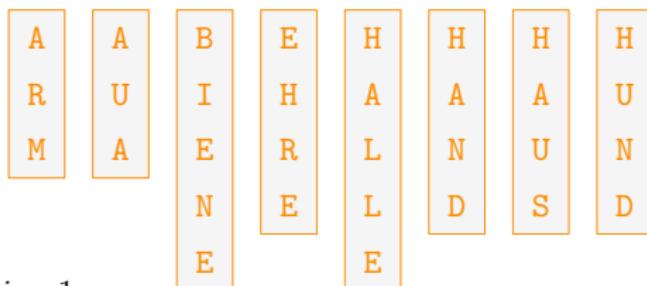
return concatenation of

(Rekursion)

mkqSort($\{e \in S : e[i] < p[i]\}$, i),

mkqSort($\{e \in S : e[i] = p[i]\}$, $i + 1$),

mkqSort($\{e \in S : e[i] > p[i]\}$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

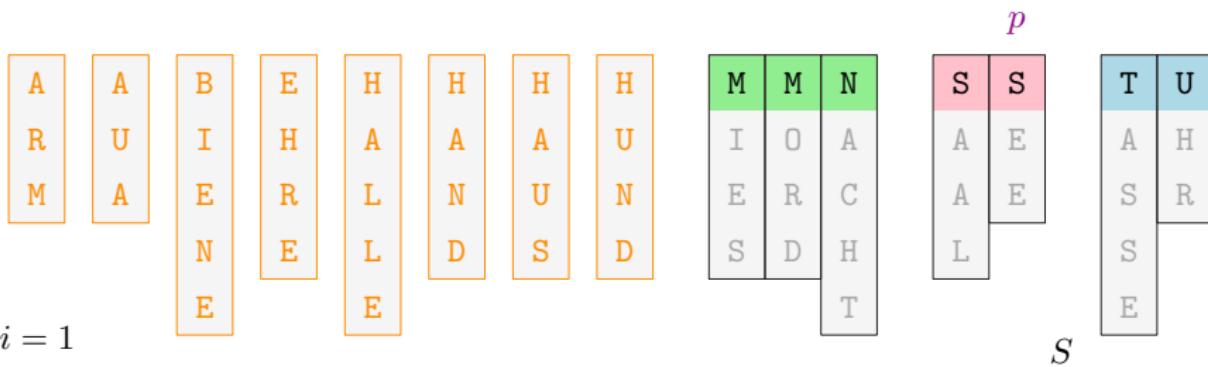
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


$i = 1$

in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

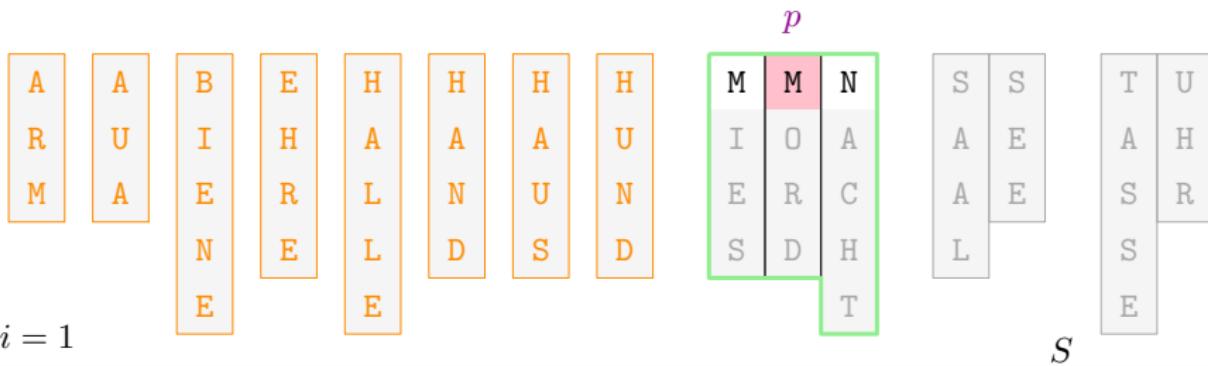
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

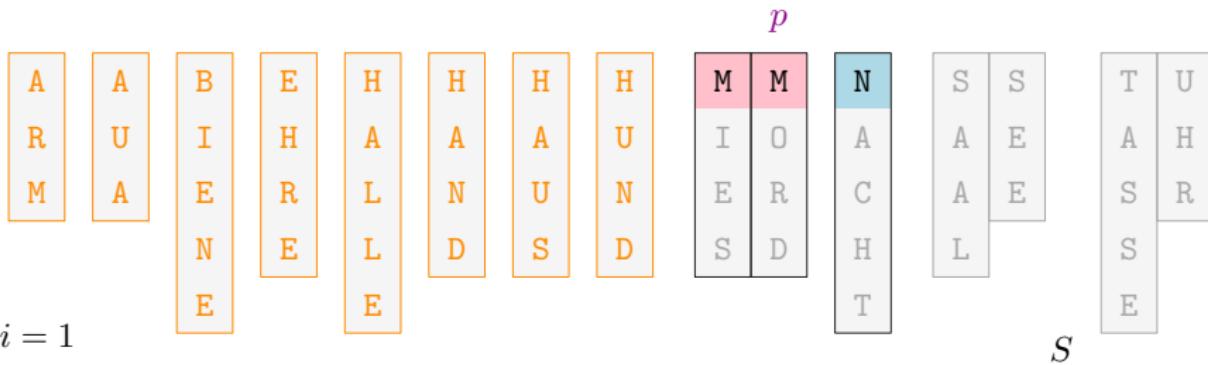
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),

mkqSort($\langle e \in S : e[i] = p[i] \rangle, i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

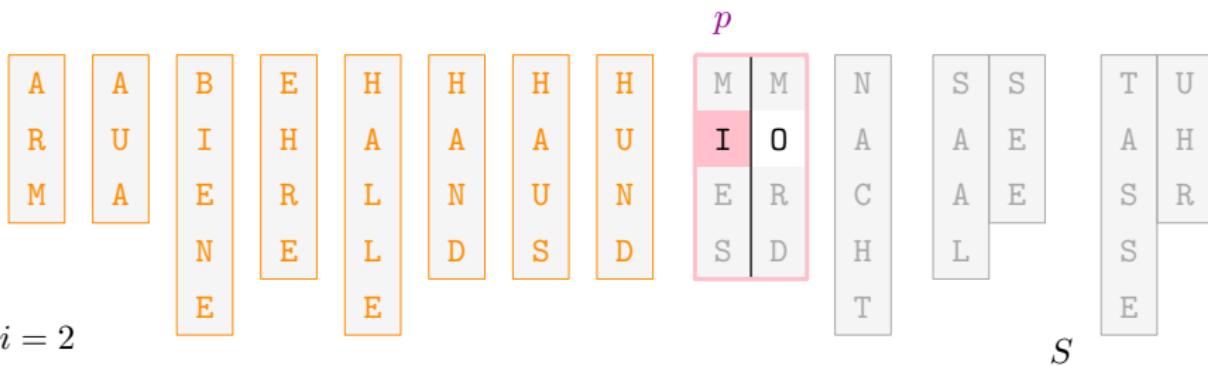
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

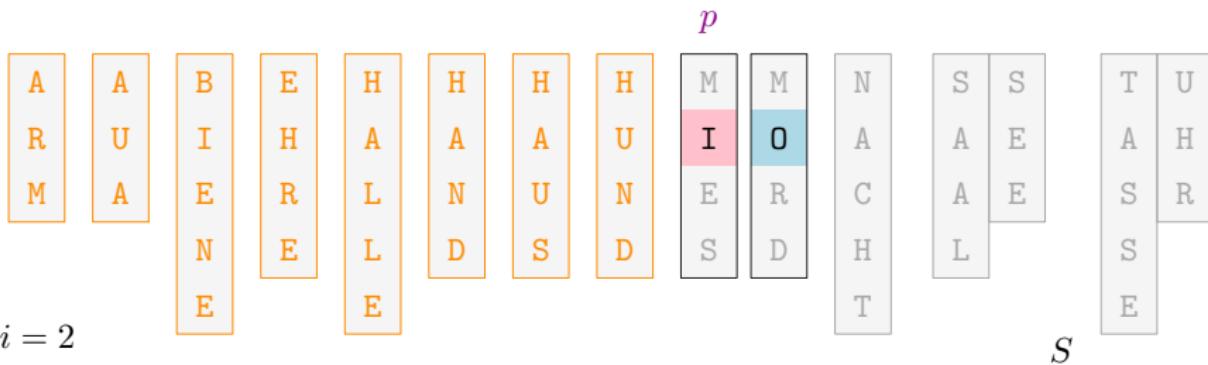
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

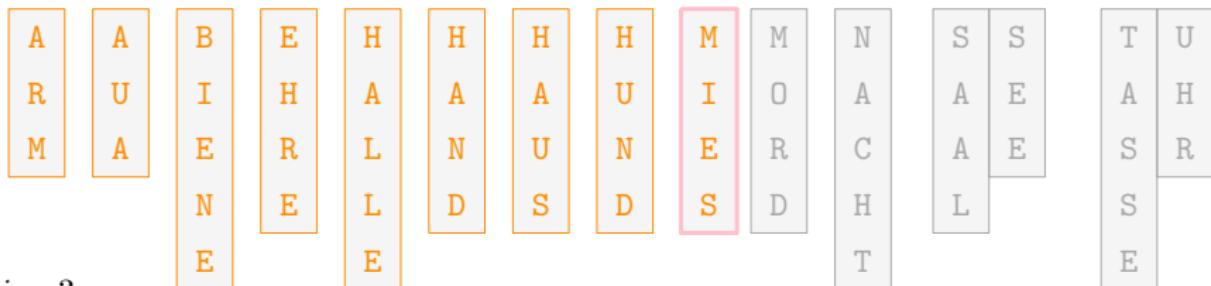
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

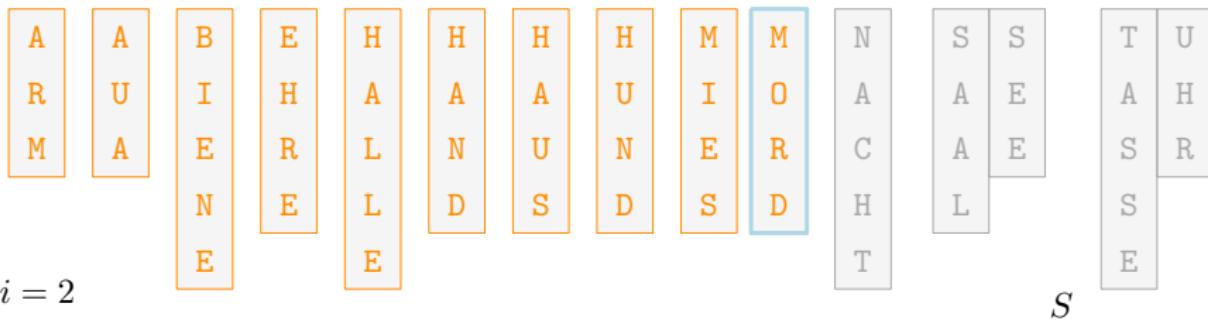
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

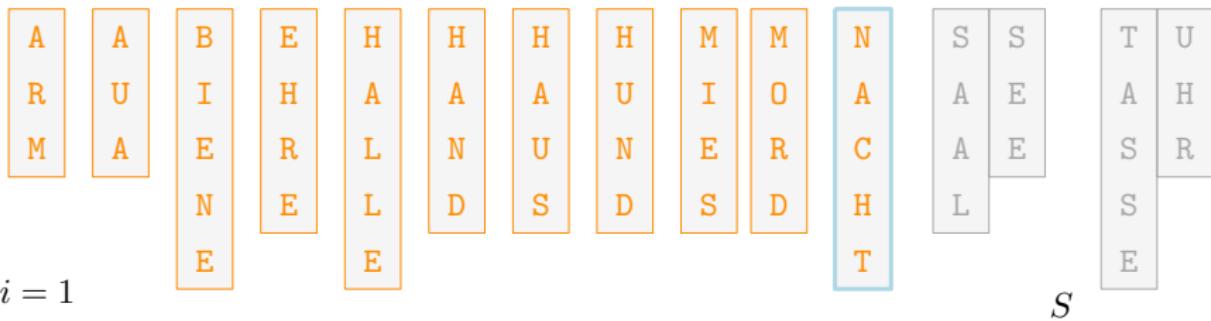
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

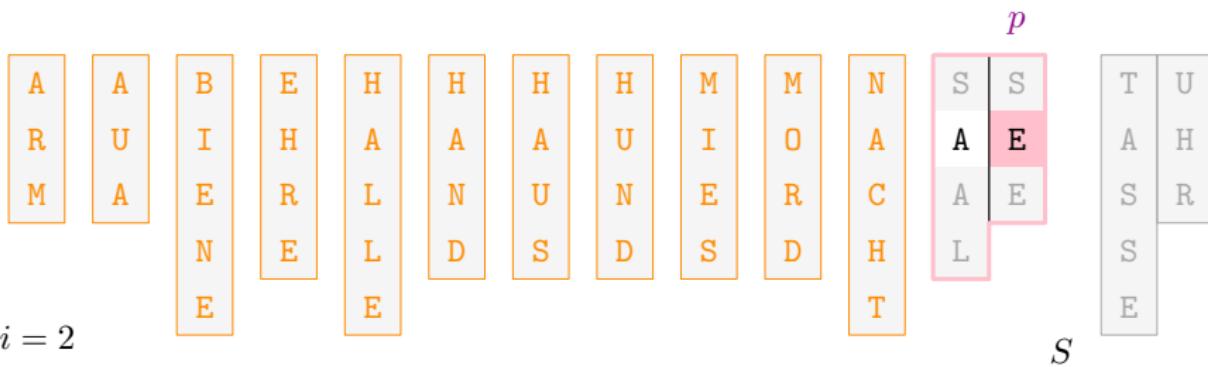
return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

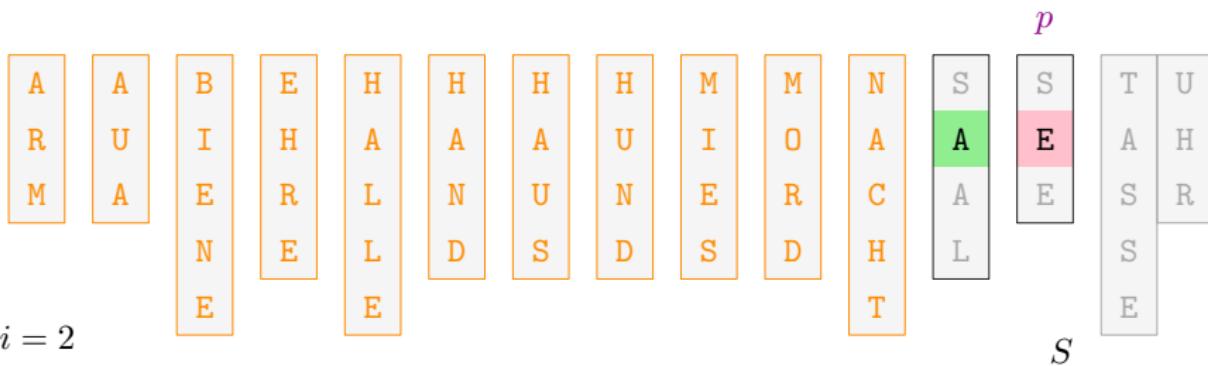
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

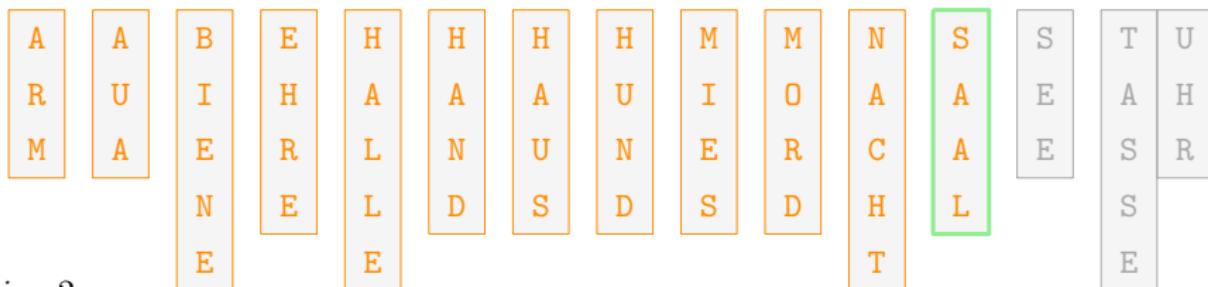
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

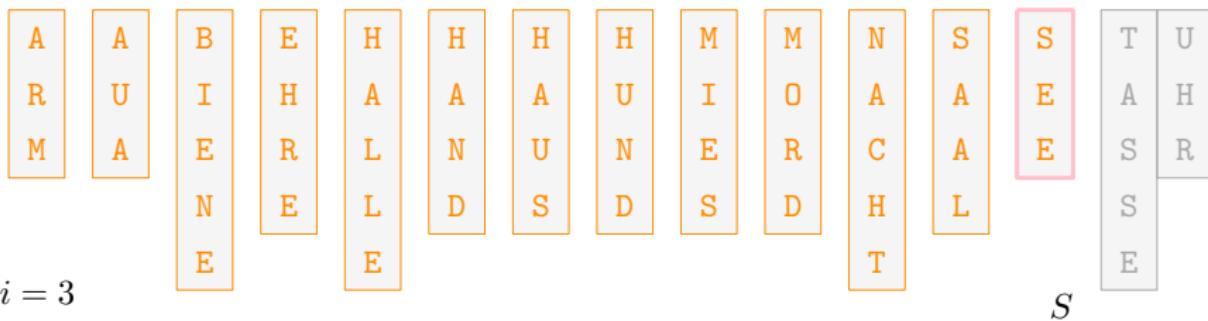
choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
mkqSort($\langle e \in S : e[i] = p[i] \rangle, i + 1$),
mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

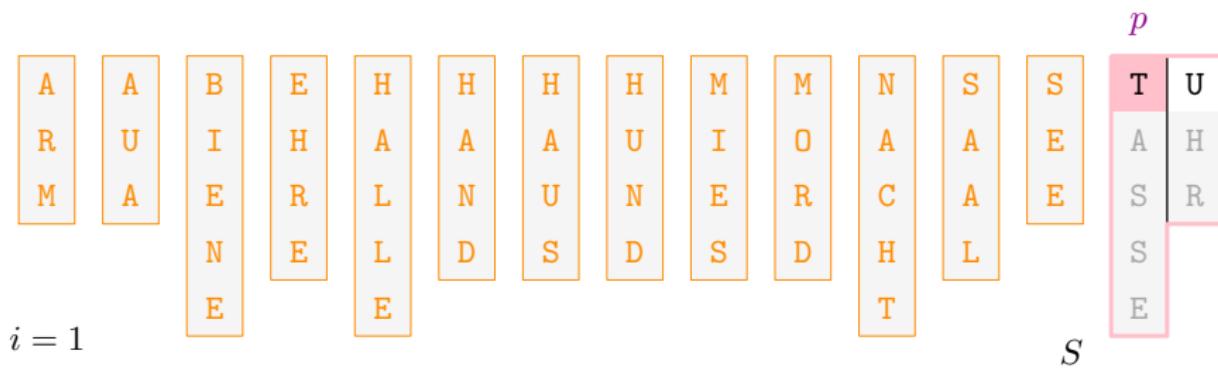
(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

 $\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$
 $\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$
 $\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$


in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
mkqSort($\langle e \in S : e[i] = p[i] \rangle, i + 1$),
mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



in-place Multikey Quicksort

Ablauf – vollständig

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

(Basisfall)

choose $p \in S$ uniformly at random

(Pivotelement)

return concatenation of

(Rekursion)

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

